

**NAME**

OMS – Oregon Micro Systems motor controllers

**DESCRIPTION**

spec supports most models of the Oregon Micro Systems motor controllers, including ISA, PCI, VME, Ethernet and serial models. spec includes support for both the encoder and the servo-motor options available with most of the cards.

The recognized ISA models are the PCX, PC34, PC38, PC39, PC46, PC48 and PC58. Both the PCI models PCIx and MAXp are recognized and supported. Modern spec configurations control all these cards completely from user level (no IRQ and no kernel driver is needed).

For VME crates, the VME8, VME44, VME58 and MAXv models are supported. The OMS VME cards are operated in polled mode in all currently recommended configurations, so no VME IRQs will be used.

spec also supports the MAXnet model over both RS-232C and Ethernet interfaces.

**DEVICE CONFIGURATION**

The OMS motor controllers are selected in the spec *config* file by lines similar to

```
PC_OMS = /dev/oms00 4 INTR
PC_OMSP = 0x330 4
PC_OMSP58 = 0x300 0xe000 4
PC_OMSV = 0xfc00 8 POLL
PC_OMSV58 = 0xe000 8 POLL
PC_OMSMAXV = 0xd000 8 POLL
PC_OMSPCI = 1 4
PC_OMSPCIM = 1 8
HW_OMSMAXN = 192.168.0.4:23 5
HW_OMSMAXN_RS = /dev/ttyS0 115200 5
```

The **Device** screen of the *edconf* program (normally run from the *config* macro) is used to select the controller type and to assign the parameters. The above selection would be displayed as follows:

Motor and Counter Device Configuration (Not CAMAC)

MOTORS	DEVICE	ADDR	<>MODE	NUM	<>TYPE
YES	/dev/oms00		INTR	4	OMS PCX/34/38/39/48 (driver)
YES		0x330		4	OMS PCX/34/38/39/48 (no driver)
YES	0xe000	0x300		4	Oregon Micro Systems PC58
YES		0xfc00	POLL	8	Oregon Micro Systems VME8/44
YES		0xe000	POLL	8	Oregon Micro Systems VME58
YES		0xd000	POLL	8	Oregon Micro Systems VME MAXv
YES		1		4	Oregon Micro Systems PCIx
YES		1		8	Oregon Micro Systems PCI MAXp
YES	/dev/ttyS0	<>	115200	5	OMS MAXnet (Serial)
YES	maxnet:23			5	OMS MAXnet (Socket)

The first example is only of historical interest. It selects the PC board with the driver node */dev/oms00*. The special kernel-level driver could be used in either interrupt or polled mode. Interrupts would be generated when motors completed their motions or hit a limit.

The second example selects one of the ISA card models with I/O port polling, with the board's base address at 0x330 and with four motors on the board.

The third example selects the PC58 board at I/O port 0x300. The PC58 also requires 4,096 bytes of low memory. In the example, the address entered as 0xE000 in the DEVICE column of the configuration editor selects a real memory address of 0xE0000. (The value in the

configuration is multiplied by 16.)

The fourth example selects the VME8 and VME44 modules, with the board's A16 base address jumpered at 0xFC00 and with the VME interrupt disabled, i.e., the board is used in polled mode. *spec*'s early VME implementation included support for VME controllers with UNIX drivers that included IRQ support. The currently active VME support in *spec* doesn't use IRQs, so all VME modules should be configured for polled mode.

The next two examples select the VME58 and MAXv models, respectively. Note, both models requires 4,096 bytes of A16 address space, so valid addresses have one hexadecimal digit followed by three zeroes. Note also, *spec* currently only supports A16 addressing for the MAXv, so make sure the card is jumpered appropriately.

The next two examples select the PCIx and the MAXp models. Each OMS PCI board should be set to a unique board address from 1 to 8, and that is the value to be entered in the ADDR field.

The second to last example is for the MAXnet controller over the serial interface. The default baud rate of the device is 115200, and that is what is shown. The last example is for the MAXnet controller using its Ethernet interface. The DEVICE column contains either an IP address or a resolvable host name. The port number can optionally follow using a colon as delimiter. If missing, the default port, 23, will be used.

## MOTOR CONFIGURATION

On the **Motor** screen of the configuration editor, all of the OMS models use one of the symbols OMS, OMS\_E, OMS\_P, OMS\_S or OMS\_M in the controller field of the screen. The OMS\_E choice indicates the motor is being used with an encoder. The OMS\_P option is for a stepping motor with an encoder and which will use the position-maintenance feature (as of *spec* release 5.08.02-1). The OMS\_S configures a DC servo motor and also enables the encoder features. The OMS\_M choice indicates the motor channel is being used with special multiplexing hardware. Contact CSS for more information on the multiplexing.

OMS motor controllers can have from two to eight motors. *spec* numbers the motors that OMS designates as X, Y, Z, T, U, V, R and S from 0 through 7, in that order.

By default, motors are automatically assigned to the controller channels in the same order as they appear on the **Motor** screen. If there is more than one controller, the channels are assigned to the controllers in the order in which the controllers appear on the **Device** screen.

Alternatively, the controller unit and motor channel numbers can be assignment explicitly on the second line of the **Motor** screen in any order. Channel numbers start at zero. Unit numbers are automatically assigned to each OMS controller configured on the **Device** screen in the order in which they appear, starting with unit zero. All OMS controller types are included in a single unit numbering sequence, and the unit numbering for OMS controllers is independent of the unit numbering for any other configured controller types.

## EMERGENCY STOP

For the MAX model controllers with firmware 1.21 or greater, *spec* will treat the case where both limits are set simultaneously as a special event. (Other OMS models do not have a command to read the status of both limit switches.) *spec* will generate an emergency stop when that happens. As is the normal case with hitting a limit, *spec* will stop all other active motors and reset to command level. For an emergency stop, a different message is printed on the screen and, if in server mode, an "emergency\_stop" event is sent to clients.

## INITIALIZATION SEQUENCE

If the nonstandard optional motor parameter "init\_sequence" is set in the *config* file for any particular motor, *spec* will send that string during hardware initialization (as of *spec* release 5.08.03-8). (Non-standard optional parameters are set from the configuration editor

by typing a `p` from the motor screen.) The string can be made up of a sequence of the following commands:

ABH	- Auxiliary bit on (fw >= 1.30)
ABL	- Auxiliary bit off (fw >= 1.30)
AF	- Auxiliary bit off (legacy)
AN	- Auxiliary bit on (legacy)
BD $x$	- Set direction of general purpose I/O bits
BH $d$	- Bit high
BL $d$	- Bit low
BS $x$	- Bit set
DBI	- Invert step direction bit
DBN	- Normal step direction bit
HH	- Home high (legacy)
HL	- Home low (legacy)
HTH	- Home active high (fw >= 1.30)
HTL	- Home active low (fw >= 1.30)
IO $d, t$	- I/O bit direction, $t = 0$ or $1$
LF	- Limits off (legacy)
LH	- Limits high (legacy)
LL	- Limits low (legacy)
LMF	- Limits off (fw >= 1.30)
LMH	- Limits on (fw >= 1.30)
LN	- Limits on (legacy)
LTH	- Limits high (fw >= 1.30)
LTL	- Limits low (fw >= 1.30)
PA $t$	- Power automatic, $t = 0$ or $1$
PH	- Auxiliary bit high (legacy)
PL	- Auxiliary bit low (legacy)
SEV	- Settling time in PA mode, $0 \leq v \leq 1000$

where  $x$  is a hexadecimal value and  $d$  is a decimal value, each with a maximum value associated with the number of general purpose I/O bits. If there are 8, the maximum value for  $x$  is FF, and the maximum value for  $d$  is 7. If there are 16 I/O bits, the maximum values are FFFF and 15, respectively. Note, though, SPEC doesn't check if the values are in range.

The commands labeled "legacy" are deprecated by OMS for "MAX" controllers with firmware versions at 1.30 or greater. The commands labeled "fw >= 1.30" can be used with such controllers.

The commands should be placed in a string, delimited by semicolons. For example, the *config* file might contain entries like these:

```
MOTPAR:init_sequence = LH;AF
MOTPAR:init_sequence = LH;PA1;SE250;BDFF00;BH1;BL2
MOTPAR:init_sequence = IO2,1;IO3,1;BH2;BH3
```

Consult the OMS manual for details on the commands.

Not all commands are available on all models. Bad commands or bad values may produce unexpected errors from the controller. The string value may be examined with `motor_par()`. Assigning a value with `motor_par()` will have no effect, as the parameter is only sent to the controller when the *config* file is read during hardware initialization.

## HOME SEARCH

Home and limit searches are started through `spec`'s built-in `chg_dial()` function and `home` macro. The motor speed of the home search is determined from the home-speed parameters, if set, otherwise from the normal base and slew rates and acceleration. (The home-speed parameters can be set on the optional motor-parameter screen of the configuration editor.)

The `chg_dial()` arguments "home+", "home-", "lim+" and "lim-" will use the commands HM, HR, JG+N and JG-N, respectively, to perform the search. Here, *N* is the home slew rate if set, otherwise the normal slew rate. If the optional home-position third argument to `chg_dial()` is present, the controller will mark the point where the home switch is triggered or where the motor stops when hitting the limit as the home position.

The `chg_dial()` argument "home" is treated differently, though. By default, the behavior associated with "home+" or "home-" will take place, depending on whether the current dial position is less than or greater than the home position, if set, or zero if not.

However, the home and limit search support for the OMS controllers was greatly expanded in `spec` release 5.05.02-7. If the optional motor parameter, "home\_method" is set and is a string, `spec` will use that string as the actual home command sequence. The string must be a list of valid OMS commands, although a parameter substitution syntax is available for the step speeds and the home position.

OMS commands that should be used in a home-method string include:

- HS – Enable home-switch mode
- HE – Enable encoder home mode
- HL – Set home switch active low
- HH – Set home switch active high
- JG – Jog, specify direction and speed
- HM – find home in forward direction
- HR – find home in reverse direction

Use of other commands may cause problems with `spec`'s communication with the controller. `spec` will include the proper axis addresses and terminators. Multiple commands should be separated by a semicolon.

The possible parameter substitution strings are:

- \$S – Base rate (slow)
- \$F – Slew rate (fast)
- \$A – Acceleration time
- \$H – Home position

Example home sequences follow:

HS;JG+1000;VL200;HL;HR0 – Enable home switch mode, run clockwise at 1000 steps/second until hitting the limit switch, set the home switch active low, set the speed to 200 steps/second, search for the home switch in the counter-clockwise direction and set the position to zero when the home switch is reached.

HE;JG+1000;VL200;HL;HR0 – As above, but enable the encoder-index home mode.

HS;JG+1000;HL;HR0;VL200;HH;HM0 – Enable home switch mode, run clockwise at 1000 steps/second until hitting the limit switch, set the home switch active low and search for it in the counter-clockwise direction, set the speed to 200 steps/second, make the home switch active high, move counter clockwise until the switch is released, and set that position as zero.

HS;JG+\$F;HL;HR0;VL2\$S;HH;HM\$H – As above, but the speeds and final home position are parameterized.

The sense of the home switch on the OMS controllers is set by software. The default sense is for the switch to be active low. The command pass through feature, described below, can change the sense. For example,

```
motor_par(tth, "send", "HH")
```

will send the "HH" command to the controller to make the home switch for the tth motor active high. If the sense needs to be changed, a startup command file could define the config\_mac macro to include code such as

```
{
  local i
  for (i = 0; i < MOTORS; i++) {
    if (substr(motor_par(i, "controller"), 1, 3) == "OMS")
      if (motor_par(i, "responsive"))
        motor_par(i, "send", "HH")
  }
}
```

The above will send the "HH" command to each OMS motor that passed the presence test. The config\_mac macro is run automatically after hardware is (re)initialized. Alternatively, the "init\_sequence" feature, described above, can be used as of spec release 5.08.03-8.

## POSITION MAINTENANCE

In order to enable position-maintenance mode on encoder-equipped stepping motors, in addition to setting the controller type to OMS\_P, several additional optional parameters need to be set. The standard optional parameter "encoder\_step\_size" must be set. Its value is the number of encoder steps per degree (or per unit), in line with the value of the standard "step\_size" parameter, which is the number of steps per degree (or per unit) as used by the positioning commands. In addition, the following non-standard optional parameters should be set:

hold\_velocity – (OMS command HV) The maximum velocity to be used when correcting position error. Note, the controller's factory default value is zero. A non-zero value is required for position maintenance to be enabled.

hold\_gain – (OMS command HG) The hold gain parameter, which is multiplied by the position error to determine the velocity used during correction.

hold\_deadband (OMS command HD) – The number of encoder counts from the target position in which no position correction is made.

See the OMS user manuals for additional details on these commands.

The non-standard optional parameters are set from the configuration editor by typing a p from the motor screen. For non-standard optional parameters, both the name of the parameter and the value are entered.

When position maintenance is enabled, spec will program the encoder ratio (OMS command ER) using the values of the "step\_size" and "encoder\_step\_size" parameters. spec will also program the hold velocity, gain and deadband if the parameters have been set. Finally, spec will enable position maintenance by sending the OMS command HN before the first move following hardware initialization and on the first moves following a limit-switch event or a move abort.

By default, spec will turn off position maintenance when exiting. To override that default behavior, set the nonstandard optional parameter "keep\_pos\_maint" to a nonzero value.

The parameter can be a controller parameter which will set values for all motors on the controller, or a motor parameter which will only set values for the individual motor. The motor parameter will override the controller parameter setting.

Note, do not select position maintenance for a motor on which there is no encoder. The software cannot detect the error in configuration, and sensible motor behavior is not guaranteed.

### SLIP DETECTION

If the non-standard optional parameter "slip\_steps" is set in the *config* file, spec will enable slip detection for stepper motors with encoders. If the controller detects motor slip, spec will treat that notification as a motor fault, which will cause spec to halt all moving motors and print an error message.

### SPECIAL COMMANDS

The following special commands are available through the `motor_par()` function. The two letter commands are direct implementations of commands described in the OMS manual. Refer to that manual for more information. Not all commands are available on all versions of the OMS controllers or on all firmware versions for a particular controller.

`motor_par(motor, "PA", mode)` – If *mode* is 1, the controller turns motor power on before each move and off after the move (assuming motor power is controlled by the auxiliary output pins). If *mode* is 0, motor power stays on.

`motor_par(motor, "SE", msec)` – Sets the settling time in milliseconds to be used before the power is reduced in PA mode.

`motor_par(motor, "AF")` – Turns auxiliary power off.

`motor_par(motor, "AN")` – Turns auxiliary power on.

`motor_par(motor, "BH", bit_num)` – Sets general purpose output pin numbered *bit\_num* high.

`motor_par(motor, "BL", bit_num)` – Sets general purpose output pin numbered *bit\_num* low.

`motor_par(motor, "BX")` – Returns the state of the general purpose input pins. A one in any binary position in the value returned indicates that the corresponding pin is low.

`motor_par(motor, "RB")` – Returns the direction of the general purpose I/O lines as bits in an integer. Output bits are a one, while input bits are a zero.

### COMMAND PASS THROUGH

Command pass through is available using the following functions. Command pass through should be used with caution to avoid interfering with the built-in programming commands spec sends to the OMS controllers.

`motor_par(motor, "send", cmd)` - Sends the string *cmd* to the OMS channel associated with *motor*. For example, set *cmd* to "LF" to disable hardware limits on the associated motor.

`motor_par(motor, "read", cmd)` - Sends the string *cmd* to the OMS channel associated with *motor*, as above, and returns a string containing the response. For example,

```
print motor_par(tth, "read", "RP")
240000
```

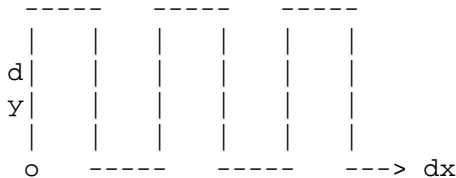
results in the string "AX RP\n" being sent to the controller.

`motor_par(motor, "usend", cmd)` - Sends the *cmd* to the OMS controller associated with *motor*. Unlike the "send" option above, no channel address or terminator characters are added to the string.

`motor_par(motor, "uread", cmd)` - Sends the string `cmd` to the OMS controller associated with `motor`, as above, and returns a string containing the response. Again, unlike the "read" option above, no channel address or terminator characters are added to the string.

### ASYNCHRONOUS SURFACE SCANNING

The following commands implement a special asynchronous, two-dimensional scanning mode. The purpose of the scan is to allow averaging of a signal scattered from different regions of a surface. The scan is in the form of a repeating square wave, as illustrated below.



The scan starts at the point `o`, as specified with the commands below, and continues in the  $x$  and  $y$  directions in the range as specified with the commands below. At the end of the range, the motors are returned to the starting position and the scan is repeated.

Two motors must be configured with the mnemonics `dx` and `dy` in order for the asynchronous scanning mode to be available. When not in scanning mode, these motors may be moved normally.

Once started, the scanning will continue until explicitly stopped either with the `stop_scan` command (shown below), with a `^C` typed at the keyboard or with a `sync` command (which aborts the motors, but doesn't update `spec`'s positions). While scanning, the `wait()` function will not indicate these motors are moving. The `getangles` command will, however, return the current positions of these motors.

When the `dx` and `dy` motors are scanning, the remaining motors may be moved independently.

`motor_par(motor, "x_start", value)` - Sets the starting position for the `dx` motor.

`motor_par(motor, "x_range", value)` - Sets the extent of the motion in the  $x$  direction.

`motor_par(motor, "x_stepsize", value)` - Sets the size of each step in  $x$ . The number of steps is determined by dividing this number into the range for  $x$ .

`motor_par(motor, "y_start", value)` - Sets the starting position for the `dy` motor.

`motor_par(motor, "y_range", value)` - Sets the extent of the motion in the  $y$  direction.

`motor_par(motor, "start_scan")` - Starts the asynchronous scan.

`motor_par(motor, "stop_scan")` - Stops the asynchronous scan.

### SEE ALSO

[www.omsmotion.com](http://www.omsmotion.com)